

Sum 04 Hyatt

## CS435/535 Networking/Distributed computing

Text:

Unix Network Programming, Vol 1 Stevens

Topics:

1. Introduction. TCP/UDP protocols. Connections, ports, windows/buffers, etc.
2. Sockets. Socket address structure. Byte order. System calls: `socket()`, `connect()`, `bind()`, `listen()`, `accept()`, `read()`, `write()`, `send()`, `recv()`. Assignment 1: write a simple C server program that will listen for a connection. write a simple client program that will connect to the server and send a single message and receive a single response that can be validated. Make certain that both ends receive exactly what they are supposed to receive, by checking inside the program, using `strcmp()`. You must use the name "client" for your client executable, and you must use the name "server" for your server executable. This will be true for the entire course, to make it easier to recognize hung processes.
3. Advanced server topics. System calls: `select()`, `FDSET()`, `FDISSET()`, `FDZERO()`, etc. Show how `select` can be used to handle connections to multiple clients at the same time from a single server process. Assignment: modify the server program previously written to communicate to multiple clients at the same time. The server should now become a simple 'echo' program in that anything it receives, it should echo right back (adding some identifying string so that you can be sure it was echoed properly). Then modify the client so that it can handle I/O to both `stdin/stdout` and the socket used to communicate with the server. To test this program you should start the server, then connect multiple clients to it, using different windows on your Unix workstation. You should be able to enter a line of text for any client and immediately see it echoed back to you by the server, proving that you are handling the `select()` call properly.
4. Multi-threaded servers. System calls: `fork()`, `signal()`, etc. Modify the server above so that it now spawns a new process to handle each connection as it is established, leaving the original server process with nothing to do but loop on the `accept()` system call. This server should handle the `SIGCHLD` signal correctly.
5. More system calls: `exec()`, `dup2()`. Modify the server code so that it executes the command sent from the client. It should handle multiple commands, one at a time. It should be able to handle command line arguments properly (ie I should be able to type `"/bin/ls -l filename"` or `"/bin/ps -ef"` and have it executed properly).
6. UDP protocol. sending/receiving messages. broadcast facility. error recovery. Assignment: write a UDP client/server and then test them to find out what is required to cause data loss. In the comments of the client and the server, give a one-paragraph explanation of how you caused the loss of UDP packets. Write a UDP server that will handle a "broadcast" so that a client can find out where the server is running without knowing in advance. The client should send a broadcast to any server running, the server should respond, then the client and server should exchange a pair of messages using normal UDP I/O (non-broadcast).
7. RPC. Introduction to `RPCGEN`, and the simple specification language it requires to build a skeleton client and server source code. RPC versions, service registration, calling conventions, etc. Assignment: modify the server assignment to become an RPC service

instead. Modify the client so that it will connect to this RPC service and behave exactly as it did when using pure TCP/IP.

8. Distributed computing. explanation of dividing work into multiple pieces that can be done in parallel. Explanation of how this can be done using TCP/IP and/or RPC programs. Assignment: take a computationally intensive numerical application and distribute it over multiple machines to make it run faster. The application will be described in class.

### Grading policy

1. There will be two tests given, a mid-term and final. These will count approximately 2/3 of your final grade.

2. You will be given several programming assignments to prepare you for the final project. The programming projects will count approximately 1/3 of your final grade. All programming assignments are individual works, and must not be copied from other students. Any copying of any kind will result in a grade of F for the entire course. Failure to turn in any programming assignment will also result in a course grade of F.

3. In unusual cases, where there is merit, the percentage of the grades given above will be varied. IE if you do poorly on the first exam, but you do well on the last exam and on the final project, then the 1/3-1/3-1/3 split will be modified somewhat to reduce the influence of the bad exam grade.

4. Programs are due on the date given in class. If a program is not turned in, you will receive a grade of F for the course. If a program is turned in late, it will be accepted but the grade for that program will never exceed a grade of "C". If you have a valid reason for failing to meet the deadline, then it is up to you to stop by my office or call me on the phone to let me know. We can work out some sort of solution, but only if you let me know before the deadline, not after it has passed.

5. If you miss an exam without notifying me, you will receive a grade of zero for that exam, which will effectively cause you to fail the course. If you can not be present for a valid reason, proof will be required. IE if you are sick, I will need proof of that in order to schedule you a make-up exam. Note that I do not give the same exam as a make-up, so you take the risk of having to answer questions that might be harder than the questions on the original exam. I don't do that intentionally, but it is difficult to make two different exams that are equal in difficulty. Which means that you are taking a risk to use this option.

6. Cheating of any kind will not be tolerated. If you cheat on exams, or if you copy/plagiarize programming assignments, you will receive a grade of F for the course and you will be permanently suspended from the University of Alabama at Birmingham. There are no exceptions to this rule.